PDS: In these notes we started using the notation
W.ω to refer to "our" island ω rather than the island indices. (A world contains
a tuple of islands.)

Recall that last time we ended with Pitts and Stark's awkward example:

$\tau = (1 \longrightarrow 1) \longrightarrow \text{int}$

$v_1 = \lambda f.(f(); 1)$

$e_2 = \text{let } x = \text{ref } 0 \text{ in } \lambda f.(x := 1; f(); !x).$

We had an intuition that the only way to access the local
state is to call the exported function f. Before f is called the
first time, x gets set to 1. And x never goes from 1 to 0.
A way of formalizing this intuition is as a very simple
state transition system:

$$(x \hookrightarrow 0) \quad \longrightarrow \quad (x \hookrightarrow 1)$$

This is a useful example: It illustrates a limitation of Pitts and Stark's original model.

History: Derek started with this line of work while trying to verify
simple and common examples with ML modules. Awkward boils down
the kind of problems he ran into when applying techniques like Pitts and
Stark's.

—

The main difference from our previous model: The definition of islands.

Before (Pitts and Stark):

$\text{Island} = \text{Sub}(\text{Heap} \times \text{Heap})$

Now (Dreyer, Neis, Birkedal, 2010):

$\text{Island} = \{ \iota=(S,c,H) \mid S \in \text{STS} \wedge c \in S.\text{States}$
$\quad \wedge\ H \in S.\text{States} \longrightarrow \text{Sub}(\text{Heap} \times \text{Heap}) \}$

$\text{STS} = \{ (\text{States},\varphi) \mid \text{States} \in \text{Set} \wedge \varphi \subseteq \text{States} \times \text{States} \wedge \varphi \text{ preorder} \}$

Idea:

S = some set of states

$c \in S$ = the current state

H = state-indexed family of heap invariants

Recall preorders are reflexive and transitive.

We want the future world
relation to be reflexive and transitive and part of moving from world
to world may be making transitions in islands.

World extension:

> $W' = (j',w') \wedge W = (j,w) \wedge j' \leq j \wedge$
>
> > $w' \in Island^{m} \wedge w \in Island^{n} \wedge m \geq n \wedge$
> >
> > $\forall i \in 1..n.\ w'_i \sqsupseteq w_i$

where

> $(S',c',H') \sqsupseteq (S,c,H)$ if
>
> > $S' = S \wedge$
> >
> > $H' = H \wedge$
> >
> > $(c,c') \in S.\varphi$

Beta/Deepak: Is it ever the case that an island wants to lose
control of some piece of the heap? If so, can it regain control
later?

> Answer: I thought the answer was no, in this language.
> We're reasoning about ML, a very weak type system
> for verification purposes. If you lose control of something,
> there's no way of getting it back.
> As opposed to a more advanced language with a linear
> type system or a type system supporting separation-logic
> style reasoning. There you need control over the STS.
> You can prevent other pieces of the program from making
> transitions.

As we set things up, it's a very simple notion of transition.
Not only do we play by these rules, but the whole program does.
In our example, we are providing this function f to the rest of
the program. The context might install its own islands that use f.
Thus when the example calls f(), some arbitrary transition might happen
in *our* transition system. Thus, our environment/context are allowed
to make the same set of transitions.

World satisfaction must change:
$h_1,h_2 : (S,c,H)$ if $(h_1,h_2) \in H(c)$
$h_1,h_2 : W$ if

> $W = (j,\omega) \wedge \omega \in Island^n \wedge$
>
> $\forall i \in \{1,2\}.\ \exists h^1_i, \cdots, h^n_i.$
>
> > $(h_i = h^1_i \uplus \cdots \uplus h^n_i \wedge$
> >
> > $\forall k \in 1..n.\ h_1^k,h_2^k : \omega.k).$

That is, the heaps can be split into little pieces satisfying the islands
and corresponding little pieces satisfy the islands (ie, the relation
at the current state of an island).

Note you can embed the previous model as a special case where each
island has one state (and so one heap relation).

We have to adjust V[ref]ρ:

V[ref]ρ = { (W,ell_1,ell_2) | ∃i. W.ω.i = (({0}, [(0,0)]), 0,
λ_.{ ([ell_1↦n],[ell_2↦n]) | n ∈ $\mathbb{Z}$ } } }

We've turned the island required by our last model's V[ref]
into a single-state island.

Recall the rant about ML-style refs. Our simple model does
not permit you to reason about ownership transfer of refs.
What you really want is to enrich the type system with substructural
state (or something more sophisticated) so that you can build
a nice model. (The ref island in Dreyer-Neis-Birkedal supports
an ownership transfer example, but it's ugly and the ugliness
exists only to support such examples.)

Why would you need anything more sophisticated than ML-style refs?
Consider trying to reason about shared state in the presence of multiple
threads.

That's it: The rest of the model is unchanged. We've simply enriched the
structure
of logical worlds.

— Soundness?

How does this change affect adequacy, the compatibility properties,
monotonicity, and
so on?
We now need to prove that world extension is transitive. We can, since the
state
transition functions in islands are transitive.
Otherwise, nothing else in the metatheory cared about the structure of the
worlds.

Where this change matters is in actually trying to prove things.

Let's return to Awkward

$\tau = (1 \longrightarrow 1) \longrightarrow \text{int}$

$v_1 = \lambda f.(f(); 1)$

$e_2 = \text{let } x = \text{ref } 0 \text{ in } \lambda f.(x := 1; f(); !x).$

Proof:

Let $W_0 = (j,\omega_1,\cdots,\omega\_\{n-1\}) \in \text{World}$.

TS: $(W_0,v_1,e_2) \in E[\tau]$.

Suppose $(W_0,K_1,K_2) \in K[\tau]$.

TS: $(W_0,K_1[v_1],K_2[e_2]) \in O$

$\iff$ $h_1;K_1[v_1] \downarrow\downarrow W_0.j\ h_2; K_2[e_2]$

Let ell $\notin \text{dom}(h_2)$.

STS: $\exists v_2.\ h_1; K_1[v_1] \downarrow\downarrow W_0.j\ h_2[\text{ell}\mapsto 0]; K_2[v_2]$.

(Note we could not decrement the step counter
as not both sides took steps.)

Idea: We've added some state to the world.
So we want to extend the world with a new island
representing

$(x \hookrightarrow 0) \quad \longrightarrow \quad (x \hookrightarrow 1).$

Define $W = W_0{+}{+}\omega$

where $\omega = (\text{sts}, 0, H)$

and $\text{sts} = (\{s_0,s_1\}, \{(s_0,s_1)\}^*)$

and $H(c) = \{ (\varnothing,[\text{ell}\mapsto 0]) \}$ if $c = s_0$

$H(c) = \{ (\varnothing,[\text{ell}\mapsto 1]) \}$ if $c = s_1$.

(Notation: r^* means the reflexive, transitive closure
of r.)

Cleary (by definition), $W \sqsupseteq W_0$. We've simply added an island.

Clearly $h_1,h_2[\text{ell}\mapsto 0] : W$.

Most of the islands except the one we just added
can be split since $h_1,h_2{:}W_0$. Our new little piece satisfies
H(0).

By monotonicity,

STS: $(W,v_1,v_2) \in V[\tau]$.

Suppose $W' \sqsupseteq W$ and $(W',v'_1,v'_2) \in V[1 \longrightarrow 1]$.

TS: $(W',(v'_1(); 1),(\text{ell} := 1; v'_2(); !\text{ell})) \in E[\text{int}]$.

Suppose $(W',K'_1,K'_2) \in K[int]$ and $h'_1,h'_2:W'$.
TS: $h'_1; K'_1[v'_1(); 1] \downarrow\downarrow W'.j\ h'_2; K'_2[ell := 1; v'_2(); !ell]$.

We don't know the code for v'_1 and v'_2. We can't β reduce them.
But we do know they're related and on the right-hand side, we can take one reduction step.

STS: $h'_1; K'_1[v'_1(); 1] \downarrow\downarrow W'.j\ h'_2[ell:=1]; K'_2[v'_2(); !ell]$.

We're in a very similar state to where we had been. We want to show there is a future world. It's time to move to the $(x \hookrightarrow 1)$ state.

Define $W'' = W'[\omega := 1]$.
(Note: This notation simply means we're changing the state of the island we just added to s_1.)
Clearly $W'' \sqsupseteq W'$: It's always a valid move to transition to s_1.
Clearly $h'_1,h'_2[ell:=1] : W''$: Again obvious; we haven't changed any other islands.

We've shown the heaps related in W''. The continuations continue to be related in W''. We could aruge it STS $(v'_1(); 1)$ and $(v'_2(); !ell)$ in W'', but proving expressions related immediately introduces a new set of continuations. We have a set of continuations; let's use them.

Define $K''_1 = K'_1[\bullet;1]$
and $K''_2 = K'_2[\bullet;!ell]$.
Observe that by monotonicty, $(W'',v'_1,v'_2) \in V[1\longrightarrow 1]$.
By compatibility, $(W'',v'_1(),v'_2()) \in E[1]$.
Thus,
STS: $(W'',K''_1,K''_2) \in K[1]$.

Suppose $W''' \sqsupseteq W''$ and $h''_1,h''_2 : W'''$.
TS: $h''_1; K''_1[()] \downarrow\downarrow W'''.j\ h''_2; K''_2[()]$
$\iff$ $h''_1; K'_1[(); 1] \downarrow\downarrow W'''.j\ h''_2; K'_2[(); !ell]$

(Worth noticing: We've had these step indices.
But since we're not doing recursion, there has been absolutely no reason to fuss with them.)

WK: $h''_1; K'_1[(); 1] \longmapsto h''_1; K'_1[1]$.

We have $W''' \sqsupseteq W'' = W'[\omega:=s_1]$.
Since there are no transitions out of s_1,
WK: $W'''[\omega] = s_1$

$\Rightarrow$ $h''_2(ell) = 1$.
$\Rightarrow$ $h''_2; K'_2[(); !ell] \mapsto h''_2; K'_2[1]$.

STS: $h''_1; K'_1[1] \downarrow\downarrow W'''.j\ h''_2; K'_2[1]$.

But the heaps are related, the continuations are related,
and the values are related. We're done.

Q.E.D.

This was a rather mechanical proof. The important bit was when
we got to the unkown functions v'_1 and v'_2, we had to use the
logical relation, quantifying over a future world they may have
moved to. The clever bit was discovering our island and its
STS.

Question: Consider the example where f : ref $\rightarrow$ unit and
we pass it f x in e_2. What goes wrong in this proof?

Answer: If you try to do the proof, you'll see exactly what's wrong.
The proof attempt will generate a counterexample.
The proof would require us to show that (ell,ell') $\in$ V[ref].
This gives us the fixed ref invariant that ell,ell' both go to
the same integer. Clearly that doesn't work.

Recall from last time. There are more extensional ways of defining the
ref invariant. You could say two locations are related at the ref type under W
if for any heaps satisfying W, those locations in those heaps have the same
integer.
Ie, when you have compatibility for ! and :=.

Aside: that's the last (or next to last) time Derek will go over such proofs
in gory detail.

—

We'll finish up with one of Derek's early motivating examples.
A lot of those examples involve a mix of local state and abstract types.

We'll define a module with two abstract types.

Both modules are defined in the same way. One checks
a dynamic condition. We want to know that the dynamic condition
is true, making them equivalent. (Think of assert statements.)

What do they do? Think of them as generating fresh symbols (α symbols and β symbols). We'll call them red and blue. Internally, symbols are represented as integers.

Twin Abstraction Example (Ahmed-Dreyer-Rossberg '09):

$\tau = \exists\alpha,\beta.\ (1\longrightarrow\alpha) \times (1\longrightarrow\beta) \times (\alpha \times \beta \longrightarrow \text{bool})$

e_1 = let x = ref 0 in pack[int,int,
    (λ_. ++x, λx.++x, λ<a,b>. a = b)] as τ

e_2 = let x = ref 0 in pack[int,int,
    (λ_. ++x, λx.++x, λ<a,b>. false)] as τ.

Intuition: Proving the equivalence is a lot like asserting a = b in the code.
The point is every time you generate and α or a β, you know it's fresh.
In particular, it's not equal to any past α or β and will never be used again.
Each integer is used at most once, for either an α or a β, but not both.

How do we formalize this?

This use of the transition system is much like "ghost states" in Hoare logic.
The physical state is not enough to deduce that the relational interpretations of α and β
are disjoint. So we maintain some logical state. Here's the island we'll use. Note
that its STS permits us to prove $a \neq b$ but is otherwise imprecise, simplifying reasoning
with it.

$\omega = (S,c,H)$

$S = (\mathbb{N} \rightharpoonup \{\text{red},\text{blue}\}, \subseteq)$

$c = \emptyset$

$H(f) = \{ ([\text{ell}_1\mapsto n], [\text{ell}_2\mapsto n]) \mid n = \text{if } f=\emptyset \text{ then } 0 \text{ else } \max(\text{dom}(f)) \}$.

On transitioning we might extend f to $f[n+1\mapsto a]$.

Where $A \rightharpoonup B$ represents finite, partial maps from A to B.

The thing that's missing from this proof is the connection with the abstract types α and β. We'll cover this next time.

(Oops from last time: Derek didn't mention that candidates have to be monotone with respect to world extension.)

The interpretations are something like

$R\_\alpha = \{ (W,n,n) \mid W.\omega.c(n) = a \}$

R_β = { (W,n,n) | W.ω.c(n) = b }.

(We have a relational interpretations of types that vary over time.)